D REMARKS

In the office action, claims 1-15 were rejected under 35 U.S.C. 103(a) as being unpatentable over McDonald, U.S. patent number 7,159,216 (hereinafter referred to as "McDonald"), and also in view of Spoltore et al. U.S. patent application number 20040015971 (hereinafter referred to as "Spoltore"). Further, Claims 17 and 18 were rejected under 35 U.S.C. 112, sixth paragraph for using "means-plus-function" language and not meeting the prerequisites of three-prong test. Furthermore, certain instances in the specification were objected to due to informality of language. However, This has been removed and the instances of informality have been modified, as per the suggestions set forth in the 'Amendments to the Specifications' section.

D.1. Rejection under 35 USC 112

In the office action, claims 17 and 18 were rejected under 35 USC 112, sixth paragraph, for using "means plus function" language. However, claims 17 and 18 have now been cancelled without prejudice or disclaimer.

D.2. Rejection under 35 USC 103

With respect to the rejections over claims 1-15 under 35 U.S.C 103 (a), which forms the basis of obviousness, the applicant respectfully submits the arguments and explanations in rebuttal, which point out the differences between Mcdonald and Spoltore combined and the present invention.

Claim 1 recites a method for allocating a processor in a non-symmetric multiprocessor system to a thread, making a request for a specific program. The non-symmetric multiprocessor includes a plurality of general-purpose and special-purpose processors. Each special-purpose processor has a corresponding local program store, which stores a plurality of specific programs. These specific programs are required by the threads of the application program. Once the application program is compiled, an allocation request from the thread, making a request for the specific program (requested program), is received by the special-purpose processor which is free, the special-purpose processor including the requested program in local program store allocated to the requesting thread. The requesting thread is granted control to execute the requested program on the special-purpose processor. Once the requesting thread completes executing the requested program on the special-purpose processor, the requesting thread relinquishes control of the special-purpose processor. The processor that is relinquished from the control of the requesting thread is added to other free special-purpose processors to receive requests from other requesting threads to execute the specific programs.

In claim 1, the thread makes a request for a specific program that is loaded in a local program store of a special-purpose processor, whereas *McDonald and Spoltore fail to teach or suggest the concept of a specific program*. Further, according to claim 1 of the present application, once the special-purpose processor is allocated, it is under the control of the requesting thread, and the requesting thread can make a request for the execution of the requested program on the special-purpose processor (one or more times). It may be noted that unlike in a scheduler of a regular operating system, wherein

the requesting thread is a program that runs on a processor, the requesting thread of the present invention *makes a request for the execution of a complete specific program and executes the specific program on the special-purpose processor.* McDonald and Spoltore fail to teach such an execution of a separate specific program by a requesting thread on a special-purpose processor. The support for this recitation is found on page 8, lines 18-32 ; page 9, lines 1- 4 and on FIG.3.

The dependency of claim 5 has been changed to make it dependent on independent claim 1. With regard to claim 5, Mcdonald (column 11, lines 47-67; column and column 12, lines 1-3; column 12, lines25-58) teaches the concept of the assignment of a thread to a preferred CPU and a preferred node. The preferred node includes a plurality of CPUs and the memory associated with them. Further, McDonald teaches dispatching of the thread to the preferred CPU, wherein the preferred CPU and the preferred node of the thread are given a priority. The initial assignment of the thread to the preferred CPU and the preferred node is not based on the availability of the required program in the associated memory of the CPU. Mcdonald fails to teach the concept of granting control of the *special-purpose processor based on the search for the special-purpose processor with the requested program already loaded in the corresponding local program store.* The special-purpose processor is allocated to a requesting thread only after verifying whether the requested program is loaded in the corresponding local program store.

According to McDonald, if the thread finds required program not present after it is dispatched to the preferred CPU, a page fault is generated to get the required

program. In contrast, the present invention teaches the concept of *allocating a special-purpose processor only after loading the requested program in the corresponding local program store of a free special-purpose processor, in the case where the available free special-purpose processor does not have the requested program.* The support for this recitation is found on page 9, lines 19-31; page 10, lines 1-3; page 10, lines 26-31; and page 11, line 1.

A new dependent claim 19 has been added to elaborate on the step of granting control to the special-purpose processor. The method teaches the creation of a data structure with information that is relevant to the special-purpose processor, along with the requested program in the local program store. The support for this recitation is found on page 10, lines 16-18.

Additionally, a new dependent claim 20 has been added to recite a method for blocking a requesting thread and adding it to a request queue if there is no free special-purpose processor available in the multiprocessor system. The support for this recitation is found on page 9, lines 25-28

The dependency of claim 9 has been changed to make it dependent on claim 5. Claim 9 recites a method for loading the requested program of the special-purpose processor. According to the present invention, a plurality of specific programs from the plurality of special-purpose processors are evicted virtually until a special-purpose processor with enough space for loading the requested program in the local program store is identified. Once such a special-purpose processor is identified, the one or more programs are actually evicted and the requested program is loaded in the local program

and is allocated to the requesting thread. Mcdonald fails to describe the concept of a *requested program in the local program store associated with a special-purpose processor.* In contrast, it teaches the generation of a page fault and a paging mechanism exhibiting the locality of the placement to get the data, unlike the memory allocation of the present invention set forth in claim 9.

Further, Mcdonald describes the allocation of threads, based on priority and waiting time and not on the requested program. Spoltore teaches the concept of removing the last task added to a set of tasks if the maximum allowable latency of any task in the set of tasks is greater than the sum of the execution of the other tasks in the set of tasks. Therefore, the removal of the tasks, as taught by Spoltore (pages 3 and 4, paragraphs 24 and 43), is based on the *maximum allowable latency*, in contrast to the *virtual eviction of the specific programs* and the actual eviction of specific programs that is based on the memory space available for the requested program in the local program store of a special-purpose processor. Hence, Mcdonald and Spoltore when combined fail to teach the concept of allocating a special-purpose processor to a requesting thread, based on the availability of the requested program in the local program store of the special-purpose processor as et forth in the present claim 9. The support of this recitation is found on page 11, lines 1-25.

With regard to Claim 10, Mcdonald (column 15, lines 40-59) describes the assignment and removal of the executing thread, based on the priority of other threads making a request for execution. Further, McDonald fails to teach virtual eviction of the programs based on the least recently used (LRU) methodology. The methodology set

forth in claim 10 in conjunction with claim 9 evicts the programs on the basis of their past usage, while allocating a requesting thread to a special-purpose processor. Similarly, in addition to LRU, the present invention also teaches virtual eviction based on the least frequently used and first-in first-out order. In light of the arguments given above McDonald fails to teach the methodology of *virtual eviction of the local programs stores of the special-purpose processor*. The support for this recitation is found on page 12, lines 1-10.

With regard to claim 11, Spoltore (pages 3 and 4, paragraphs 42 and 43) describes the removal of the latest task from the set, based on the maximum allowable latency, and fails to teach a method for virtual eviction, based on task priority, execution, pending time, and relevance. It may be noted that Spoltore teaches the removal of a task, based on the maximum time a task can wait to get assigned. The present invention in claims 9 and 11 recites the concept of virtual eviction, based on the memory space available for the requested program in the local program store of the special-purpose processor. The support for this recitation is found on page 11, lines 9-14.

Furthermore, a new dependent claim 21 has been added to recite the concept of evicting the plurality of specific programs from the corresponding local program stores of all special-purpose processors till a special-purpose processor with memory space available for loading the requested program is identified. The support for this recitation is found on page on page 11, lines 16-23.

Claim 7 of the present invention recites a method for allocating at least one

special-purpose processor to a requesting thread of an application program in a multi-processor computing system. The special-purpose processor has access to a corresponding local program store. The local program store includes various specific programs required by the requesting thread. The specific program requested by the requesting thread is known as a requested program (specific program). The special-purpose processor receives the requesting thread for the specific program and stalls the request if it is not free. Further, the requesting thread is allocated a special-purpose processor once the special-purpose processor is free. After allocating the special-purpose processor to the requesting thread, the requested program is executed by the requesting thread on the special-purpose processor. Thereafter, the special-purpose processor is relinquished by the requesting thread after executing the requested program.

With regard to claim 7, the thread requests for a specific program loaded in a local program store of a special-purpose processor whereas McDonald and Spoltore fail to teach or suggest the concept of a specific program. Further, according to present invention, once the special-purpose processor is allocated, it is in the control of the requesting thread and the requesting thread can then request for the execution of the requested program on the special-purpose processor (one or more times). It may be noted that, unlike in a scheduler of a regular operating system, wherein the requesting thread is a program that runs on a processor, the requesting thread of the present invention requests for the execution of a complete specific program and executes the specific program on the special-purpose processor. McDonald and Spoltore fail to teach such an *execution of a separate specific program by a requesting thread on a special-*

purpose processor. Further, Spoltore teaches adding of unassigned task in a set of tasks waiting to get assigned but fails to teach a concept of adding the requesting thread requesting for a specific program in a request queue. The support for these recitations is found on page 8, lines 18-32 , page 9, lines 1- 4 and FIG.3

With regard to claim 8, Mcdonald (column 11, lines 47-67; column and column 12, lines 1-3; column 12, lines25-58) teaches the concept of the assignment of a thread to a preferred CPU and a preferred node. The preferred node includes a plurality of CPUs and the memory associated with them. Further, McDonald teaches dispatching of the thread to the preferred CPU, wherein the preferred CPU and the preferred node of the thread are given a priority. The initial assignment of the thread to the preferred CPU and the preferred node is not based on the availability of the required program in the associated memory of the CPU. Mcdonald fails to describe the concept of *allocating the special-purpose processor based on the search for the special-purpose processor with the requested program already loaded in the corresponding local program store.* The special-purpose processor is allocated to a requesting thread only after verifying whether the requested program is loaded in the corresponding local program store.

According to McDonald, if the thread finds required program not present after it is dispatched to the preferred CPU, a page fault is generated to get the required program. In contrast, the present invention teaches the concept of allocating a special-purpose processor only after loading the requested program in the corresponding local program store of a free special-purpose processor, in case the available free special-purpose processor does not have the requested program. The support for the

recitations of claim 8 is found on page 9, lines 19-31; page 10, lines 1-3; page 10, lines 26-31; and page 11, line 1.

The dependency of claim 12 has been changed so as to make it dependent on claim 8. Claim 12 teaches a method for loading the requested program of the special-purpose processor. According to the present invention, a plurality of specific programs from the plurality of special-purpose processors are evicted virtually till a special-purpose processor with enough space for loading the requested program in the local program store is identified. Once such a special-purpose processor is identified, the one or more programs are actually evicted and the requested program is loaded in the local program and is allocated to the requesting thread. Mcdonald fails to teach the concept of a requested program in the local program store associated with a special-purpose processor. In contrast, it teaches the generation of a page fault and a paging mechanism exhibiting the locality of the placement to get the data, unlike the memory allocation of the present invention.

Further, Mcdonald teaches the allocation of threads, based on priority and waiting time and not on the requested program. Spoltore teaches the concept of removing the last task added to a set of tasks if the maximum allowable latency of any task in the set of tasks is greater than the sum of the execution of the other tasks in the set of tasks. Therefore, the removal of the tasks, as taught by Spoltore (pages 3 and 4, paragraphs 24 and 43), is based on the *maximum allowable latency*, in contrast to the *virtual eviction of the specific programs and the actual eviction of specific programs* that is based on the memory space available for the requested program in the local program

store of a special-purpose processor. Hence, Mcdonald and Spoltore combined fail to teach the concept in claim 12 of allocating a special-purpose processor to a requesting thread, based on the availability of the requested program in the local program store of the special-purpose processor. The support of these recitations is found on page 11, lines 1-25.

With regard to Claim 13, Mcdonald (column 15, lines 40-59) teaches the assignment and removal of the executing thread, based on the priority of other threads making a request for execution. Further, McDonald fails to teach virtual eviction of the programs based on the least recently used (LRU) methodology. The methodology set forth in claim 13 evicts the programs on the basis of their past usage, while allocating a requesting thread to a special-purpose processor. Similarly, in addition to LRU, the present invention also teaches virtual eviction based on the least frequently used and first-in first-out order. In light of the arguments given above McDonald fails to teach the methodology of virtual eviction of the local programs stores of the special-purpose processor. The support for this recitation is found on page 12, lines 1-10.

With regard to claim 14, Spoltore (pages 3 and 4, paragraphs 42 and 43) teaches the removal of the latest task from the set, based on the maximum allowable latency, and fails to teach a method for virtual eviction, based on task priority, execution, pending time, and relevance. It should be noted that Spoltore teaches the removal of a task, based on the *maximum time a task can wait to get assigned*. The present invention is directed to the concept of *virtual eviction*, based on the memory space available for the requested program in the local program store of the special-purpose

processor. The support for this recitation is found on page 11, lines 9-14.

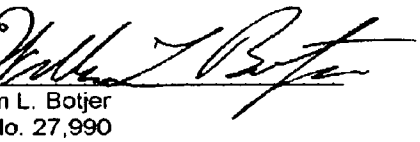
With regard to claim 15, independent claim 7 has been embodied as a computer program product. In light of the arguments given above, the applicant believes that claim 15 is now novel and non-obvious over McDonald and Spoltore.

With regard to claim 16, the system provides a compilation service for compiling an application program on an operating system. The application program comprises a plurality of interacting threads. A processor-allocation service is provided to schedule and allocate various processors, including general-purpose processors and special-purpose processors, to respective requesting threads. Further, the system provides a local program store-managing service that manages the corresponding local program stores of the special-purpose processors. The local program store-managing service also evicts and loads the requested program in the corresponding local program store of a special-purpose processor. However, Mcdonald does not teach a system for allocating a special-purpose processor to a requesting thread with a requested program. McDonald fails to teach the concept of allocation of a requested program to the requesting thread, to be executed on the special-purpose processor. Further, Spoltore (page 3, paragraphs 36 and 27) merely teaches the assignment of tasks for execution and fails to teach the concept of virtual eviction and loading of specific programs in the local program store of the special-purpose processor.

CONCLUSION

In view of the present amendment, it is submitted that the claims are patentably distinct over the cited art and that the objections to the claims and specifications have been overcome, and notice to that effect is earnestly solicited. Accordingly, reconsideration of the present application is requested. If the Examiner has any questions regarding this matter, the Examiner is requested to telephone the applicant's attorney at the numbers listed below, prior to issuing a further office action.

Respectfully submitted,

By 

William L. Botjer
Reg. No. 27,990
PO Box 478
Center Moriches, NY 11934
(212) 737-5728 (Tue-Thurs)
(631) 874-4826 (Mon & Fri)
(631) 834-0611 (cell if others busy)

Dated: November 29, 2007